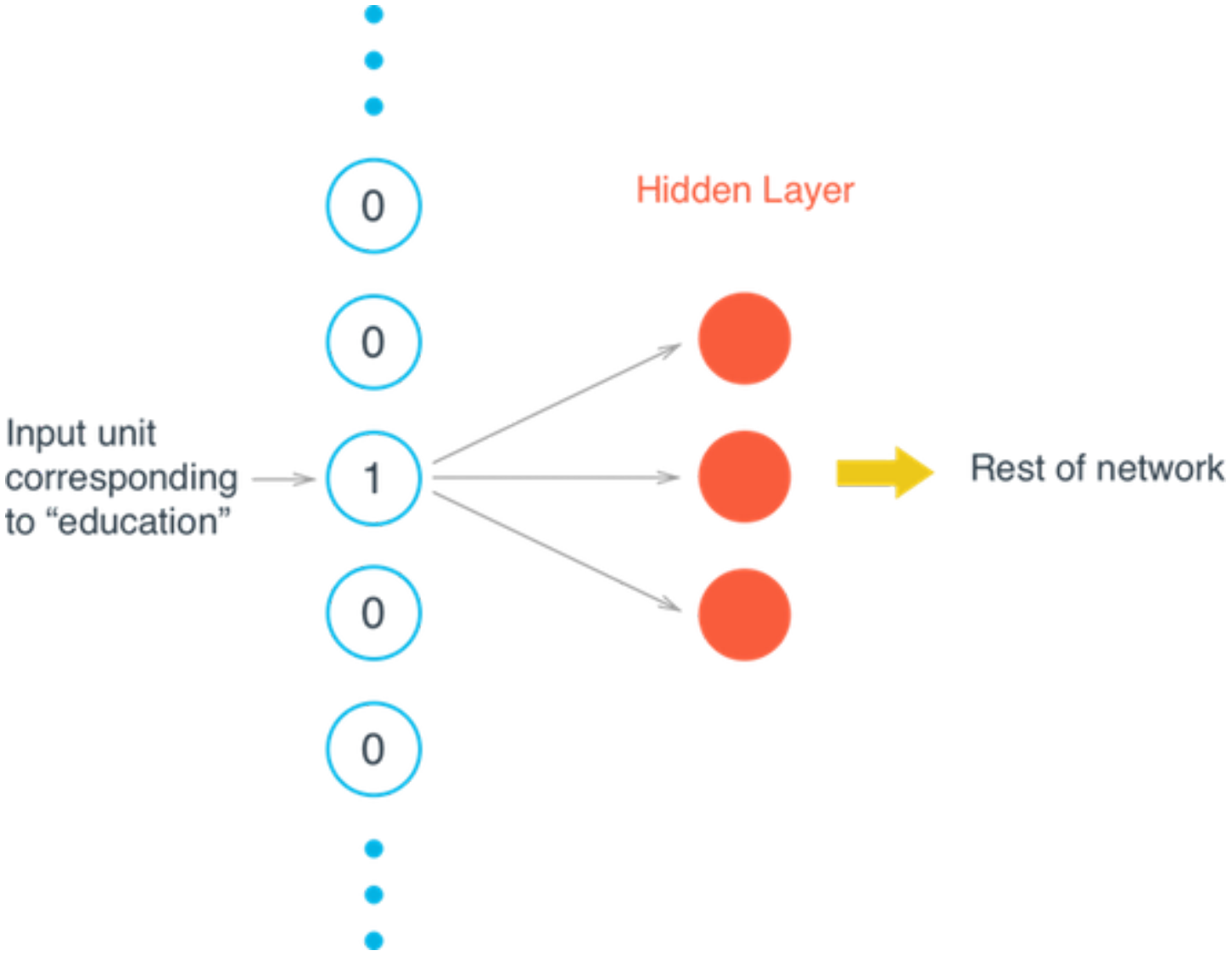


理解 Word2Vec 之 Skip-Gram 模型

原文英文文档请参考链接：

- Word2Vec Tutorial - The Skip-Gram Model
- Word2Vec (Part 1): NLP With Deep Learning with Tensorflow (Skip-gram)

什么是Word2Vec和Embeddings？

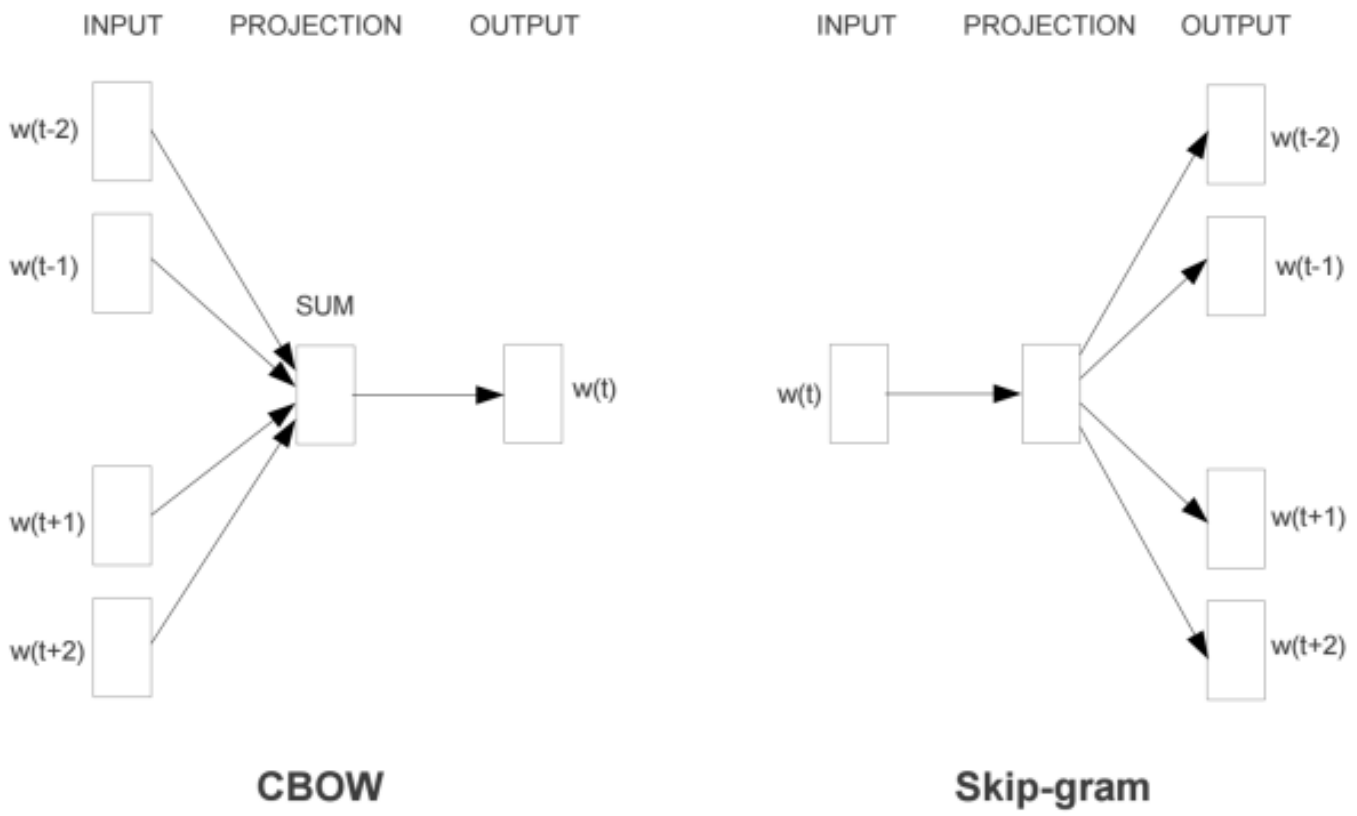
Word2Vec是从大量文本语料中以无监督的方式学习语义知识的一种模型，它被大量地用在自然语言处理（NLP）中。那么它是如何帮助我们做自然语言处理呢？Word2Vec其实就是通过学习文本来用词向量的方式表征词的语义信息，即通过一个嵌入空间使得语义上相似的单词在该空间内距离很近。Embedding其实就是一个映射，将单词从原先所属的空间映射到新的多维空间中，也就是把原先词所在空间嵌入到一个新的空间中去。

我们从直观角度上来理解一下，cat这个单词和kitten属于语义上很相近的词，而dog和kitten则不是那么相近，iphone这个单词和kitten的语义就差的更远了。通过对词汇表中单词进行这种数值表示方式的学习（也就是将单词转换为词向量），能够让我们基于这样的数值进行向量化的操作从而得到一些有趣的结论。比如说，如果我们对词向量kitten、cat以及dog执行这样的操作：kitten - cat + dog，那么最终得到的嵌入向量（embedded vector）将与puppy这个词向量十分相近。

第一部分

模型

Word2Vec模型中，主要有Skip-Gram和CBOW两种模型，从直观上理解，Skip-Gram是给定input word来预测上下文。而CBOW是给定上下文，来预测input word。本篇文章仅讲解Skip-Gram模型。



Skip-Gram模型的基础形式非常简单，为了更清楚地解释模型，我们先从最一般的基础模型来看Word2Vec（下文中所有的Word2Vec都是指Skip-Gram模型）。

Word2Vec模型实际上分为了两个部分，**第一部分为建立模型，第二部分是通过模型获取嵌入词向量。**Word2Vec的整个建模过程实际上与自编码器（auto-encoder）的思想很相似，即先基于训练数据构建一个神经网络，当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，例如隐层的权重矩阵——后面我们将会看到这些权重在Word2Vec中实际上就是我们试图去学习的“word vectors”。基于训练数据建模的过程，我们给它一个名字叫“Fake Task”，意味着建模并不是我们最终的目的。

> 上面提到的这种方法实际上会在无监督特征学习（unsupervised feature learning）中见到，最常见的就是自编码器（auto-encoder）：通过在隐层将输入进行编码压缩，继而在输出层将数据解码恢复初始状态，训练完成后，我们会将输出层“砍掉”，仅保留隐层。

The Fake Task

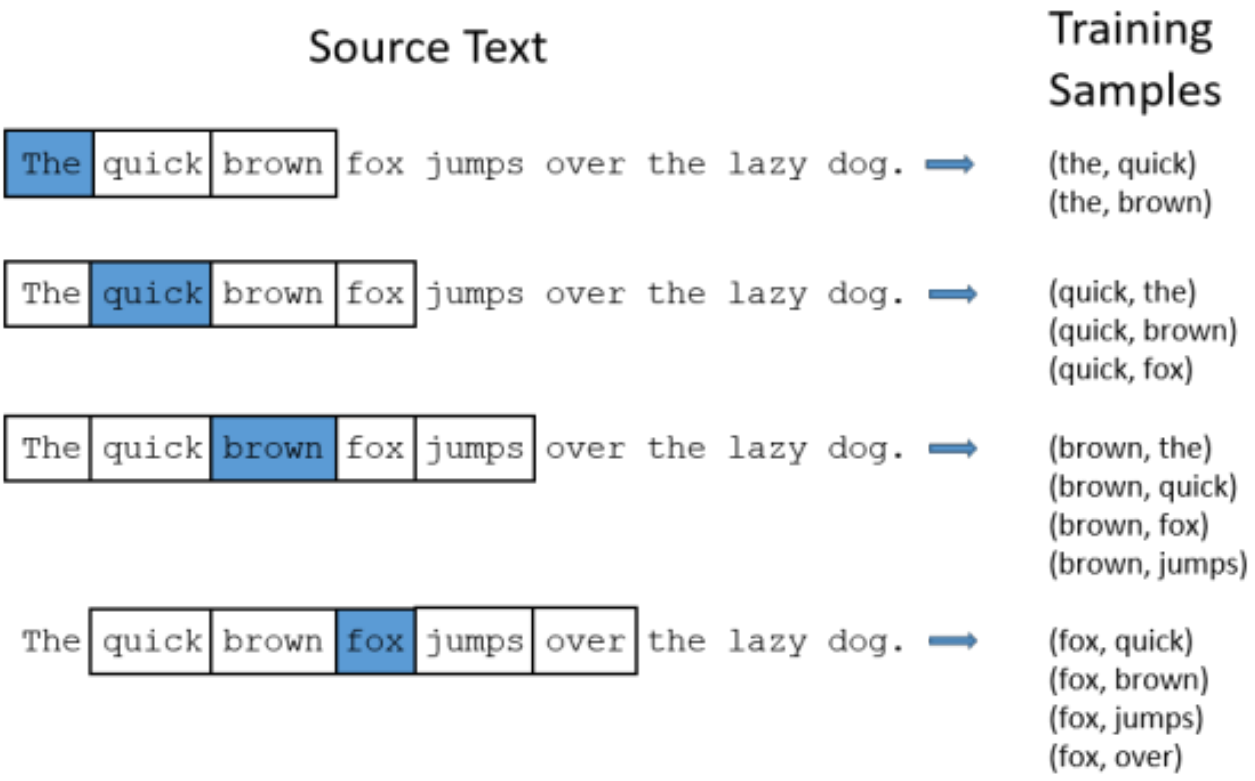
我们在上面提到，训练模型的真正目的是获得模型基于训练数据学得的隐层权重。为了得到

这些权重，我们首先要构建一个完整的神经网络作为我们的“Fake Task”，后面再返回来看通过“Fake Task”我们如何间接地得到这些词向量。

接下来我们来看看如何训练我们的神经网络。假如我们有一个句子**“The dog barked at the mailman”**。

- 首先我们选句子中间的一个词作为我们的输入词，例如我们选取“dog”作为input word；
- 有了input word以后，我们再定义一个叫做skip_window的参数，它代表着我们从当前input word的一侧（左边或右边）选取词的数量。如果我们设置 $skip\_window = 2$ ，那么我们最终获得窗口中的词（包括input word在内）就是**['The', 'dog'，'barked', 'at']**。$skip\_window = 2$ 代表着选取左input word左侧2个词和右侧2个词进入我们的窗口，所以整个窗口大小 $span = 2 \times 2 = 4$ 。另一个参数叫num_skips，它代表着我们从整个窗口中选取多少个不同的词作为我们的output word，当 $skip\_window = 2$ ，$num\_skips = 2$ 时，我们将会得到两组 **(input word, output word)** 形式的训练数据，即 **('dog', 'barked')，('dog', 'the')**。
- 神经网络基于这些训练数据将会输出一个概率分布，这个概率代表着我们的词典中的每个词是output word的可能性。这句话有点绕，我们来看个栗子。第二步中我们在设置skip_window和num_skips=2的情况下获得了两组训练数据。假如我们先拿一组数据 **('dog', 'barked')** 来训练神经网络，那么模型通过学习这个训练样本，会告诉我们词汇表中每个单词是“barked”的概率大小。

模型的输出概率代表着到我们词典中每个词有多大可能性跟input word同时出现。举个栗子，如果我们向神经网络模型中输入一个单词“Soviet“，那么最终模型的输出概率中，像“Union”， ”Russia“这种相关词的概率将远高于像”watermelon“，”kangaroo“非相关词的概率。因为”Union“，”Russia“在文本中更大可能在”Soviet“的窗口中出现。我们将通过给神经网络输入文本中成对的单词来训练它完成上面所说的概率计算。下面的图中给出了一些我们的训练样本的例子。我们选定句子**“The quick brown fox jumps over lazy dog”**，设定我们的窗口大小为2（ $window\_size = 2$ ），也就是说我们仅选输入词前后各两个词和输入词进行组合。下图中，蓝色代表input word，方框内代表位于窗口内的单词。



我们的模型将会从每对单词出现的次数中习得统计结果。例如，我们的神经网络可能会得到更多类似（“Soviet“，”Union“）这样的训练样本对，而对于（”Soviet“，”Sasquatch“）这样的组合却看到的很少。因此，当我们的模型完成训练后，给定一个单词”Soviet“作为输入，输出的结果中”Union“或者”Russia“要比”Sasquatch“被赋予更高的概率。

模型细节

我们如何来表示这些单词呢？

首先，我们都知道神经网络只能接受数值输入，我们不可能把一个单词字符串作为输入，因此我们得想个办法来表示这些单词。最常用的办法就是基于训练文档来构建我们自己的词汇表（vocabulary）再对单词进行one-hot编码。

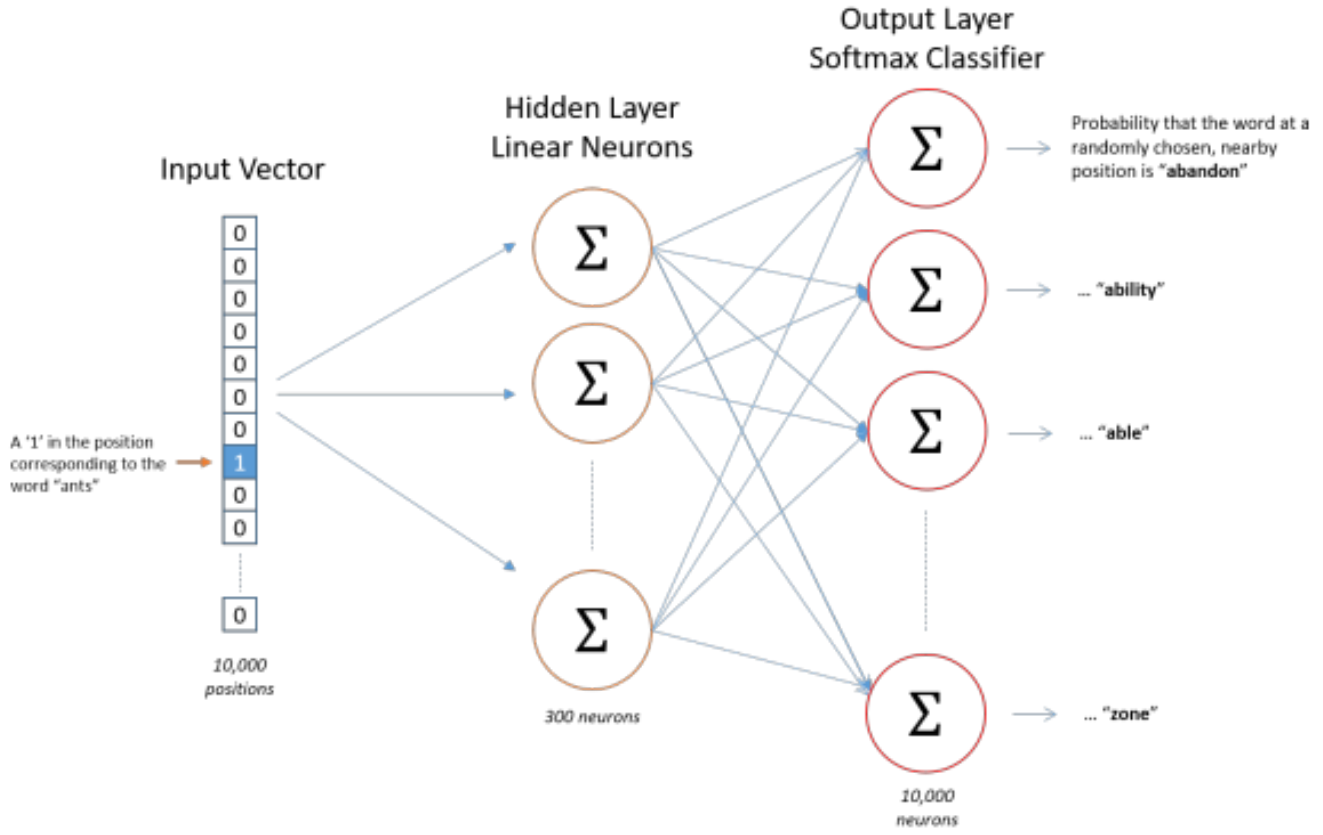
假设从我们的训练文档中抽取出10000个唯一不重复的单词组成词汇表。我们对这10000个单词进行one-hot编码，得到的每个单词都是一个10000维的向量，向量每个维度的值只有0或者1，假如单词ants在词汇表中的出现位置为第3个，那么ants的向量就是一个第三维度取值为1，其他维都为0的10000维的向量（ $ants = [0, 0, 1, 0, ..., 0]$ ）。

还是上面的例子，“The dog barked at the mailman”，那么我们基于这个句子，可以构建一个大小为5的词汇表（忽略大小写和标点符号）：("the", "dog", "barked", "at", "mailman")，我们对这个词汇表的单词进行编号0-4。那么”dog“就可以被表示为一个5维向量[0, 1, 0, 0,

0]。

模型的输入如果为一个10000维的向量，那么输出也是一个10000维度（词汇表的大小）的向量，它包含了10000个概率，每一个概率代表着当前词是输入样本中output word的概率大小。

下图是我们神经网络的结构：



隐层没有使用任何激活函数，但是输出层使用了sotfmax。

我们基于成对的单词来对神经网络进行训练，训练样本是（input word, output word）这样的单词对，input word和output word都是one-hot编码的向量。最终模型的输出是一个概率分布。

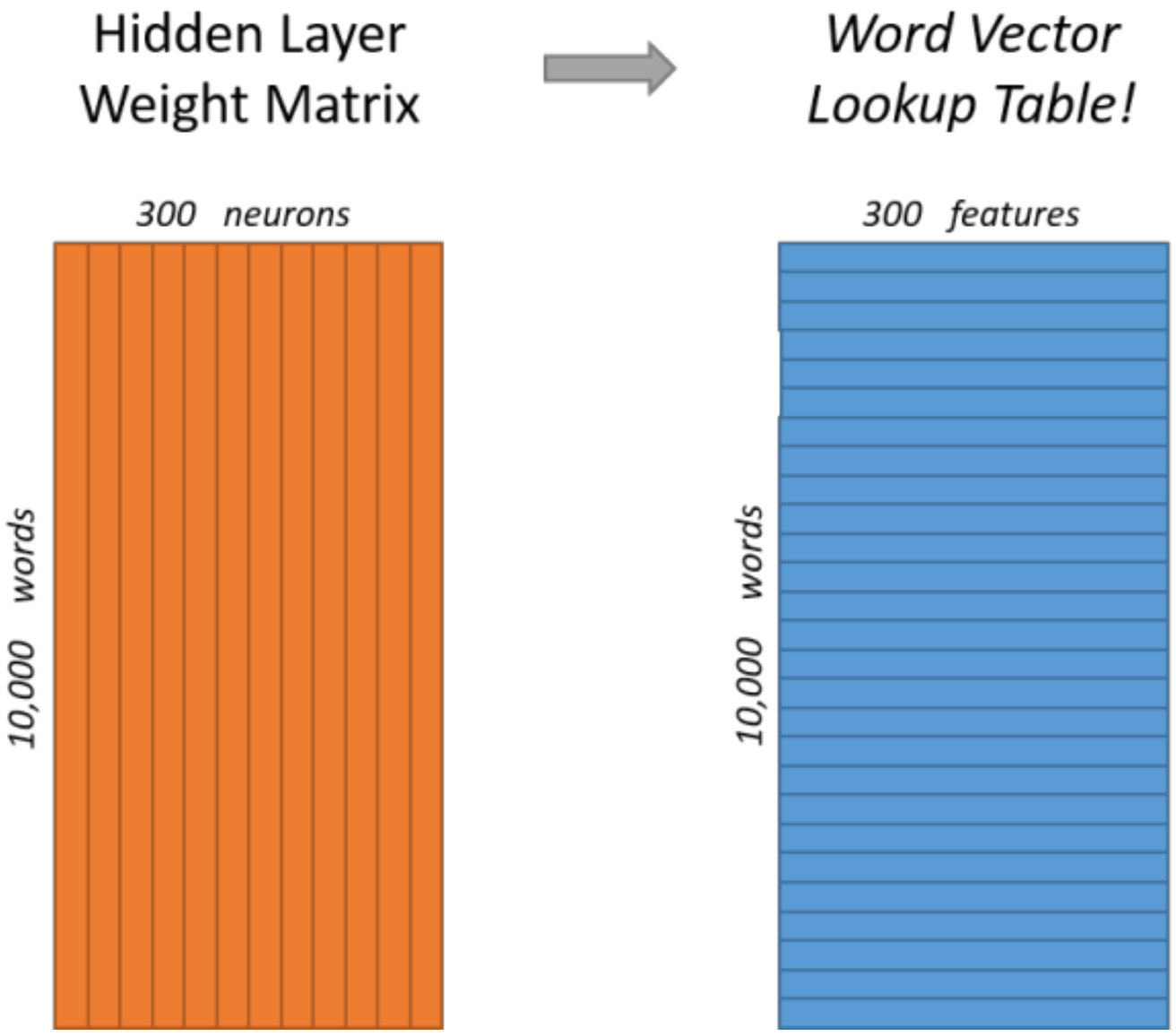
隐层

说完单词的编码和训练样本的选取，我们来看下我们的隐层。如果我们现在想用300个特征来表示一个单词（即每个词可以被表示为300维的向量）。那么隐层的权重矩阵应该为10000行，300列（隐层有300个结点）。

Google在最新发布的基于Google news数据集训练的模型中使用的就是300个特征的词向量。词向量的维度是一个可以调节的超参数（在Python的gensim包中封装的Word2Vec接口

默认的词向量大小为100， window_size为5）。

看下面的图片，左右两张图分别从不同角度代表了输入层-隐层的权重矩阵。左图中每一列代表一个10000维的词向量和隐层单个神经元连接的权重向量。从右边的图来看，每一行实际上代表了每个单词的词向量。



所以我们最终的目标就是学习这个隐层的权重矩阵。

我们现在回来接着通过模型的定义来训练我们的这个模型。

上面我们提到，input word和output word都会被我们进行one-hot编码。仔细想一下，我们的输入被one-hot编码以后大多数维度上都是0（实际上仅有一个位置为1），所以这个向量相当稀疏，那么会造成什么结果呢。如果我们将一个1 x 10000的向量和10000 x 300的矩阵相乘，它会消耗相当大的计算资源，为了高效计算，它仅仅会选择矩阵中对应的向量中维度值为1的索引行（这句话很绕），看图就明白。

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

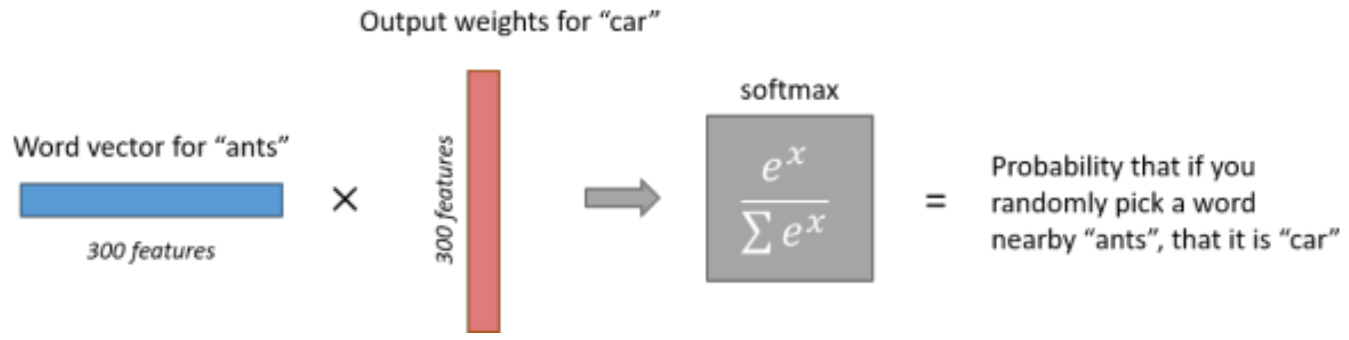
我们来看一下上图中的矩阵运算，左边分别是1 x 5和5 x 3的矩阵，结果应该是1 x 3的矩阵，按照矩阵乘法的规则，结果的第一行第一列元素为 $0 \times 17 + 0 \times 23 + 0 \times 4 + 1 \times 10 + 0 \times 11 = 10$，同理可得其余两个元素为12，19。如果10000个维度的矩阵采用这样的计算方式是十分低效的。

为了有效地进行计算，这种稀疏状态下不会进行矩阵乘法计算，可以看到矩阵的计算的结果实际上是矩阵对应的向量中值为1的索引，上面的例子中，左边向量中取值为1的对应维度为3（下标从0开始），那么计算结果就是矩阵的第3行（下标从0开始）—— [10, 12, 19]，这样模型中的隐层权重矩阵便成了一个”查找表“（lookup table），进行矩阵计算时，直接去查输入向量中取值为1的维度下对应的那些权重值。隐层的输出就是每个输入单词的“嵌入词向量”。

输出层

经过神经网络隐层的计算，ants这个词会从一个1 x 10000的向量变成1 x 300的向量，再被输入到输出层。输出层是一个softmax回归分类器，它的每个结点将会输出一个0-1之间的值（概率），这些所有输出层神经元结点的概率之和为1。

下面是一个例子，训练样本为 (input word: “ants”， output word: “car”) 的计算示意图。



直觉上的理解

下面我们将通过直觉来进行一些思考。

如果两个不同的单词有着非常相似的“上下文”（也就是窗口单词很相似，比如“Kitty climbed the tree”和“Cat climbed the tree”），那么通过我们的模型训练，这两个单词的嵌入向量将非常相似。

那么两个单词拥有相似的“上下文”到底是什么含义呢？比如对于同义词“intelligent”和“smart”，我们觉得这两个单词应该拥有相同的“上下文”。而例如”engine“和”transmission“这样相关的词语，可能也拥有着相似的上下文。

实际上，这种方法实际上也可以帮助你进行词干化（stemming），例如，神经网络对”ant“和”ants”两个单词会习得相似的词向量。

> 词干化（stemming）就是去除词缀得到词根的过程。

第二部分

第一部分我们了解skip-gram的输入层、隐层、输出层。在第二部分，会继续深入讲如何在skip-gram模型上进行高效的训练。

在第一部分讲解完成后，我们会发现Word2Vec模型是一个超级大的神经网络（权重矩阵规模非常大）。

举个栗子，我们拥有10000个单词的词汇表，我们如果想嵌入300维的词向量，那么我们的**输入-隐层权重矩阵**和**隐层-输出层的权重矩阵**都会有 10000 x 300 = 300万个权重，在如此庞大的神经网络中进行梯度下降是相当慢的。更糟糕的是，你需要大量的训练数据来调整这些权重并且避免过拟合。百万数量级的权重矩阵和亿万数量级的训练样本意味着训练这个模型将会是个灾难（太凶残了）。

Word2Vec的作者在它的第二篇论文中强调了这些问题，下面是作者在第二篇论文中的三个创新：

1. 将常见的单词组合（word pairs）或者词组作为单个“words”来处理。
2. 对高频次单词进行抽样来减少训练样本的个数。
3. 对优化目标采用“negative sampling”方法，这样每个训练样本的训练只会更新一小部分的模型权重，从而降低计算负担。

事实证明，对常用词抽样并且对优化目标采用“negative sampling”不仅降低了训练过程中的计算负担，还提高了训练的词向量的质量。

Word pairs and "phases"

论文的作者指出，一些单词组合（或者词组）的含义和拆开以后具有完全不同的意义。比如“Boston Globe”是一种报刊的名字，而单独的“Boston”和“Globe”这样单个的单词却表达不出这样的含义。因此，在文章中只要出现“Boston Globe”，我们就应该把它作为一个单独的词来生成其词向量，而不是将其拆开。同样的例子还有“New York”，“United Stated”等。

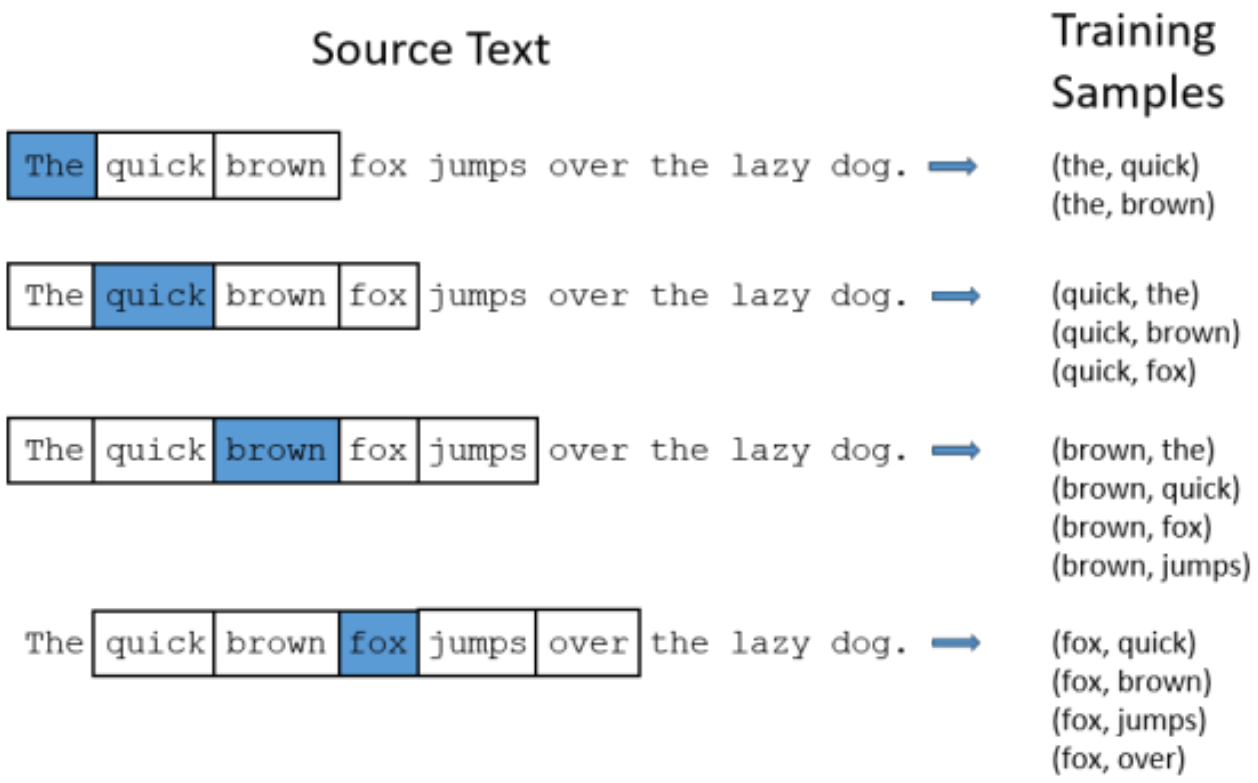
在Google发布的模型中，它本身的训练样本中有来自Google News数据集中的1000亿的单词，但是除了单个单词以外，单词组合（或词组）又有3百万之多。

如果你对模型的词汇表感兴趣，可以点击这里，你还可以直接浏览这个词汇表。

如果想了解这个模型如何进行文档中的词组抽取，可以看论文中“Learning Phrases”这一章，对应的代码word2phrase.c被发布在这里。

对高频词抽样

在第一部分的讲解中，我们展示了训练样本是如何从原始文档中生成出来的，这里我再重复一次。我们的原始文本为“The quick brown fox jumps over the laze dog”，如果我使用大小为2的窗口，那么我们可以得到图中展示的那些训练样本。



但是对于“the”这种常用高频单词，这样的处理方式会存在下面两个问题：

1. 当我们得到成对的单词训练样本时，("fox", "the") 这样的训练样本并不会给我们提供关于“fox”更多的语义信息，因为“the”在每个单词的上下文中几乎都会出现。
2. 由于在文本中“the”这样的常用词出现概率很大，因此我们将会有大量的（ ”the“，... ）这样的训练样本，而这些样本数量远远超过了我们学习“the”这个词向量所需的训练样本数。

Word2Vec通过“抽样”模式来解决这种高频词问题。它的基本思想如下：对于我们在训练原始文本中遇到的每一个单词，它们都有一定概率被我们从文本中删掉，而这个被删除的概率与单词的频率有关。
如果我们设置窗口大小 $span = 10$（即 $skip\_window = 5$ ），并且从我们的文本中删除所有的“the”，那么会有下面的结果：

1. 由于我们删除了文本中所有的“the”，那么在我们的训练样本中，“the”这个词永远也不会出现在我们的上下文窗口中。
2. 当“the”作为input word时，我们的训练样本数至少会减少10个。

> 这句话应该这么理解，假如我们的文本中仅出现了一个“the”，那么当这个“the”作为input word时，我们设置span=10，此时会得到10个训练样本 ("the", ...) ，如果删掉这个“the”，我们就会减少10个训练样本。实际中我们的文本中不止一个“the”，因此

> 当“the”作为input word的时候，至少会减少10个训练样本。

上面提到的这两个影响结果实际上就帮助我们解决了高频词带来的问题。

抽样率

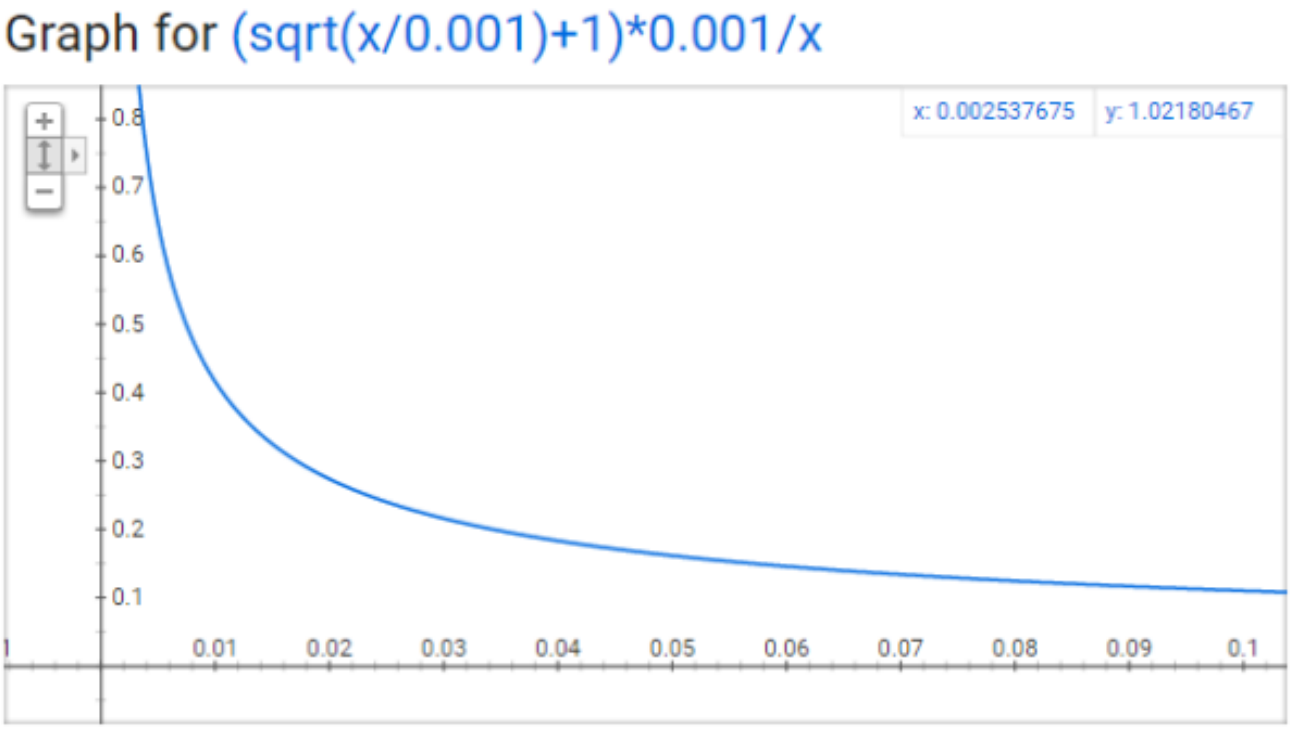
word2vec的C语言代码实现了一个计算在词汇表中保留某个词概率的公式。

w_i是一个单词，$Z(w_i)$是w_i这个单词在所有语料中出现的频次。举个栗子，如果单词“peanut”在10亿规模大小的语料中出现了1000次，那么$Z(\text{“}peanut\text{”}) = 1000/1000000000 = 1e-6$。

在代码中还有一个参数叫“sample”，这个参数代表一个阈值，默认值为0.001**（在gensim包中的Word2Vec类说明中，这个参数默认为0.001，文档中对这个参数的解释为“ threshold for configuring which higher-frequency words are randomly downsampled”）**。这个值越小意味着这个单词被保留下来的概率越小（即有越大的概率被我们删除）。

$P(w_i)$代表着保留某个单词的概率：

$$P(w_i) = (\sqrt{\frac{Z(w_i)}{0.001}} + 1) \times \frac{0.001}{Z(w_i)}$$



图中x轴代表着 $Z(w_i)$ ，即单词 w_i 在语料中出现频率，y轴代表某个单词被保留的概率。对于一个庞大的语料来说，单个单词的出现频率不会很大，即使是常用词，也不可能特别大。

从这个图中，我们可以看到，随着单词出现频率的增高，它被采样保留的概率越来越小，我们还可以看到一些有趣的结论：

- 当 $Z(w_i) <= 0.0026$ 时， $P(w_i) = 1.0$ 。当单词在语料中出现的频率小于0.0026时，它是100%被保留的，这意味着只有那些在语料中出现频率超过0.26%的单词才会被采样。
- 当 $Z(w_i) = 0.00746$ 时， $P(w_i) = 0.5$ ，意味着这一部分的单词有50%的概率被保留。
- 当 $Z(w_i) = 1.0$ 时， $P(w_i) = 0.033$ ，意味着这部分单词以3.3%的概率被保留。

> 如果你去看那篇论文的话，你会发现作者在论文中对函数公式的定义和在C语言代码的实现上有一些差别，但我认为C语言代码的公式实现是更权威的一个版本。

负采样（negative sampling）

训练一个神经网络意味着要输入训练样本并且不断调整神经元的权重，从而不断提高对目标的准确预测。每当神经网络经过一个训练样本的训练，它的权重就会进行一次调整。

正如我们上面所讨论的，vocabulary的大小决定了我们的Skip-Gram神经网络将会拥有大规模的权重矩阵，所有的这些权重需要通过我们数以亿计的训练样本来进行调整，这是非常消耗计算资源的，并且实际中训练起来会非常慢。

负采样（negative sampling）解决了这个问题，它是用来提高训练速度并且改善所得到词向量的质量的一种方法。不同于原本每个训练样本更新所有的权重，负采样每次让一个训练样本仅仅更新一小部分的权重，这样就会降低梯度下降过程中的计算量。

当我们用训练样本（input word: "fox"，output word: "quick") 来训练我们的神经网络时，“fox”和“quick”都是经过one-hot编码的。如果我们的vocabulary大小为10000时，在输出层，我们期望对应“quick”单词的那个神经元结点输出1，其余9999个都应该输出0。在这里，这9999个我们期望输出为0的神经元结点所对应的单词我们称为“negative” word。

当使用负采样时，我们将随机选择一小部分的negative words（比如选5个negative words）来更新对应的权重。我们也会对我们的“positive” word进行权重更新（在我们上面的例子

中，这个单词指的是”quick“）。

> 在论文中，作者指出指出对于小规模数据集，选择5-20个negative words会比较好，对于大规模数据集可以仅选择2-5个negative words。

回忆一下我们的隐层-输出层拥有300 x 10000的权重矩阵。如果使用了负采样的方法我们仅仅去更新我们的positive word-“quick”的和我们选择的其他5个negative words的结点对应的权重，共计6个输出神经元，相当于每次只更新 $300 \times 6 = 1800$ 个权重。对于3百万的权重来说，相当于只计算了0.06%的权重，这样计算效率就大幅度提高。

如何选择negative words

我们使用“一元模型分布（unigram distribution）”来选择“negative words”。

要注意的一点是，一个单词被选作negative sample的概率跟它出现的频次有关，出现频次越高的单词越容易被选作negative words。

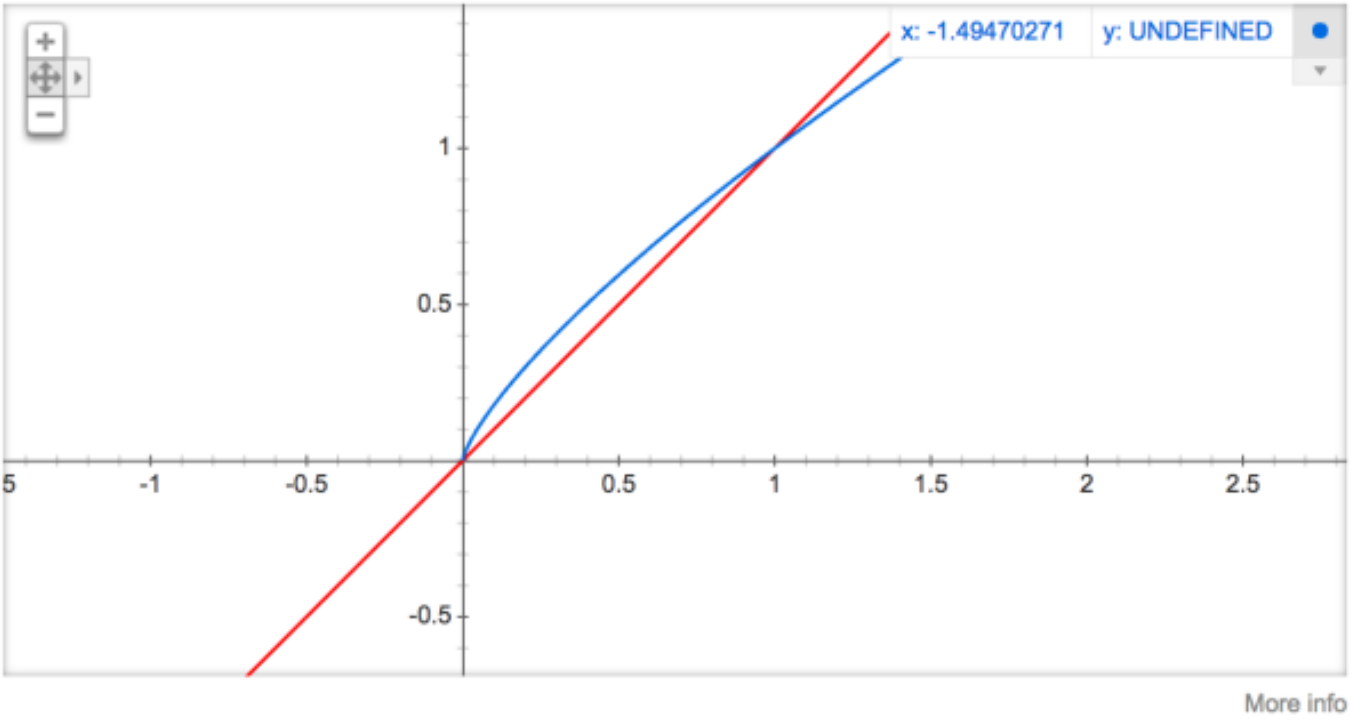
在word2vec的C语言实现中，你可以看到对于这个概率的实现公式。每个单词被选为“negative words”的概率计算公式与其出现的频次有关。

代码中的公式实现如下：

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j)^{3/4})}$$

每个单词被赋予一个权重，即 $f(w_i)$， 它代表着单词出现的频次。

公式中开3/4的根号完全是基于经验的，论文中提到这个公式的效果要比其它公式更加出色。你可以在google的搜索栏中输入“plot y = x^(3/4) and y = x”，然后看到这两幅图（如下图），仔细观察x在[0,1]区间内时y的取值， $x^{3/4}$ 有一小段弧形，取值在 $y = x$ 函数之上。



负采样的C语言实现非常的有趣。unigram table有一个包含了一亿个元素的数组，这个数组是由词汇表中每个单词的索引号填充的，并且这个数组中有重复，也就是说有些单词会出现多次。那么每个单词的索引在这个数组中出现的次数该如何决定呢，有公式 $P(w_i) * table\_size$ ，也就是说计算出的**负采样概率*1亿=单词在表中出现的次数**。

有了这张表以后，每次去我们进行负采样时，只需要在0-1亿范围内生成一个随机数，然后选择表中索引号为这个随机数的那个单词作为我们的negative word即可。一个单词的负采样概率越大，那么它在这个表中出现的次数就越多，它被选中的概率就越大。

到目前为止，Word2Vec中的Skip-Gram模型就讲完了，对于里面具体的数学公式推导细节这里并没有深入。这篇文章只是对于实现细节上的一些思想进行了阐述。

其他资料

如果想了解更多的实现细节，可以去查看C语言的实现源码。

其他Word2Vec教程请参考这里。